Lattice-Based Cryptography in Miden VM

Alan Szepieniec[1] and Frederik Vercauteren[2]

`alan@asdm.gmbh`, `frederik.vercauteren@gmail.com`

[1] AS Discrete Mathematics GmbH, Switzerland
[2] CCI, Belgium

1 Introduction

The prime number $p = 18446744069414584321 = 2^{64} - 2^{32} + 1 = \Phi_6(2^{32})$ is incredibly useful for proof systems for general-purpose computational integrity. This usefulness originates from several factors:

- integers modulo p fit into one 64-bit register;
- the field $\mathbb{Z}_p, +, \times$ has a multiplicative subgroup of order 2^{32}, giving rise to fast arithmetic using NTTs as well as the FRI [1] polynomial commitment scheme for constructing transparent SNARKs;
- any integer modulo p can be decomposed into the unique pair of 32-bit limbs and the correctness of this decomposition can be proven efficiently;
- the product of any two integers less than 2^{32} is less than p;
- the balanced ternary expansion is sparse, and thus gives rise to efficient modular reduction.

In light of these compelling properties, p was selected by a number of independent teams for their general purpose computational integrity proof systems, including Polygon Zero, Triton VM, Polygon Miden, Risc Zero, and Polygon Hermez.

It is often necessary to prove the correct execution of cryptographic algorithms, such as public key encryption or signature verification. The different cost model of proof systems versus physical computers motivates a re-evaluation of the appropriate cryptographic algorithms. This line of research gave rise to two elliptic curves, one defined by Pornin [11] over degree-five extension field of $\mathbb{Z}_p$, and one defined by Salen *et al.* [13] over a degree-six extension field of $\mathbb{Z}_p$.

This report focuses on lattice-based cryptography. Specifically:

- Section 2 addresses the question how to do the arithmetic for lattice-based cryptography efficiently on virtual machines defined over $\mathbb{Z}_p$, including NTT, memory addressing, and embedding multiple integers into each field element.
- Section 3 discusses supporting the two lattice-based signature schemes selected by the NIST PQC project, Falcon [6] and Dilithium [10].
- Section 4 proposes a new lattice-based public key encryption scheme defined over $\mathbb{Z}_p$, which makes use of the embedding technique described above.
- Section 5 proposes a *publicly re-randomizable* commitment scheme based on essentially the same construction.

Supporting python code for these techniques and proposed algorithms is available at `https://github.com/ASDiscreteMathematics/Miden_Lattices`.

[*] Date of this document: 11th August 2022.

2 Fast arithmetic in cyclotomic rings

2.1 Non-native parameter sets

The lattice based schemes Kyber, Saber, Falcon and Dilithium, which are all finalists in the NIST pqcrypto standardization effort (and many other schemes), all rely on arithmetic in cyclotomic rings of the form

$$R_{q,n} = \mathbb{Z}_q[x]/(x^n + 1)$$

for some modulus q (not necessarily prime) and $n = 2^k$, and more in particular $n = 256, 512, 1024$. The moduli used by the different schemes are as follows:

- Kyber: $q = 3329$
- Saber: $q = 2^{13}$
- Falcon: $q = 12289$
- Dilithium: $q = 2^{23} - 2^{13} + 1$

When the modulus q is chosen such that $2n|\varphi(q)$ (in many cases q is a prime so then $2n|q-1$), it is well known that arithmetic in R_q can be computed efficiently using the number theoretic transform. This can be done natively for Falcon and Dilithium, almost natively for Kyber and with a work-around for Saber.

In Miden, the native modulus is $p = 2^{64} - 2^{32} + 1$, so $\mathbb{Z}_q$ in this case contains a root of unity of order 2^{32} and in particular of order 2^k for any $k = 1, \ldots, 32$, and we therefore get fast arithmetic in $R_{p,n}$ for all such $n = 2^k$.

2.2 Bound on the size of *q*

The inequality $p \gg q$ suggests that we might simulate ring multiplication in $R_{q,n}$ using NTTs modulo p without running into problems arising from the reduction modulo p. This is indeed the case.

Since the q used in the above schemes is different from the native p, we first need to give a bound for the maximum modulus size q for each ring $R_{q,n}$ such that we can recover the product exactly (using an extra modular reduction) from the product in $R_{p,n}$.

So assume we are given two elements $a(x), b(x) \in R_{q,n}$ written as $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$, then the product $c(x) = \sum_{i=0}^{n-1} c_i x^i$ satisfies

$$c_i = \sum_{j=0}^{i} a_{i-j} b_j - \sum_{j=i+1}^{n-1} a_{n-j+i} b_j \, .$$

In particular, we have a sum of n products of elements in $\mathbb{Z}_q$ (with $\pm$), so we see that the maximum bound on c_i is given

$$|c_i| < nq^2 \, ,$$

assuming that all coefficients were represented in $[0, q)$. If a symmetric interval $[q/2, q/2)$ was used to represent elements in $R_{q,n}$ the bound is clearly sufficient as well. Since we need to be able to recover this as an integer (it can be negative) to be able to reduce modulo q afterwards, it suffices that $p \geq 2nq^2$. For popular choices of n above we therefore obtain the following upper bounds:

n	$\max \log_2(q)$
256	27.5
512	27
1024	26.5

2.3 Lazy reduction

In the schemes that use a module structure such as Kyber, Saber and Dilithium, one often has to compute a matrix vector product $\mathbf{A} \cdot \mathbf{v}$ where the matrix and vector contain elements of $R_{q,n}$. Assuming that the matrix has ℓ columns, we therefore could add ℓ such products together before doing the final reduction modulo q. The addition of ℓ such products simply introduces an extra factor of ℓ in the above bound. The largest number of columns appearing in all of the above schemes is 7 in the case of Dilithium level 5 parameter set. It is easy to verify that for this case we have $p > 7 \cdot 2 \cdot 256 \cdot q^2$, so we can indeed postpone the final reduction after the addition of the ℓ products.

2.4 Multiplication using NTT

Let N be a power of 2 and assume that $q - 1 \equiv 0 \bmod 2N$, and let ψ be a primitive $2N$-th root of unity and $\omega = \psi^2$ a primitive N-th root of unity. The N-point NTT of a sequence $[a[0], \ldots, a[N-1]]$ is denoted as $\tilde{a} = NTT(a)$ and defined by $\tilde{a}[i] = \sum_{j=0}^{N-1} a[j]\omega^{ij}$ for $i = 0, \ldots, n-1$. The inverse transformation $b = INTT(\tilde{a})$ is given by $b[i] = \frac{1}{N} \sum_{j=0}^{N-1} a[j]\omega^{-ij}$, which can be computed by replacing ω by ω^{-1} and scaling by N.

Since ω is an N-th root of unity, note that this also corresponds to the evaluation of the polynomial $a(x) \in \mathbb{Z}_q[x]/(x^N - 1)$ in all powers of ω, and as such we can multiply two polynomials in $a(x), b(x) \in \mathbb{Z}_q[x]/(x^N - 1)$ by computing

$$c(x) = INTT(NTT(a) \cdot NTT(b))$$

where $\cdot$ denotes pointwise multiplication.

The standard approach to NTT is given in Algorithm 1 where BitReverse computes an array A such that $A[k] = a[BitReverse(k)]$ obtained by reversing the binary expansion of k using $b = \log_2(N)$ bits to write k. In particular, if $k = k_0 \ldots k_{b-1}$ then $BitReverse(k) = k_{b-1} \ldots k_0$.

The above however is not directly applicable since we need arithmetic in the ring $\mathbb{Z}_q[x]/(x^N + 1)$. Note that the roots of $x^N + 1$ are given by ψ^{2k+1} for $k = 0, \ldots, n-1$, we would need to compute the evalutions of $a(x)$ in those

Algorithm 1: *Iterative NTT*

```
Input: Polynomial $a(x) \in \mathbb{Z}_q[\mathbf{x}]$ of degree $N-1$ and $N$-th primitive root $\omega_N \in \mathbb{Z}_q$ of unity
Output: Polynomial $A(x) \in \mathbb{Z}_q[\mathbf{x}] = \mathrm{NTT}(a)$
begin
    $A \leftarrow BitReverse(a)$;
    for $m = 2$ to $N$ by $m = 2m$ do
        $\omega_m \leftarrow \omega_N^{N/m}$ ;
        $\omega \leftarrow 1$ ;
        for $j = 0$ to $m/2 - 1$ do
            for $k = 0$ to $N - 1$ by $m$ do
                $t \leftarrow \omega \cdot A[k + j + m/2]$ ;
                $u \leftarrow A[k + j]$ ;
                $A[k + j] \leftarrow u + t \bmod q$ ;
                $A[k + j + m/2] \leftarrow u - t \bmod q$ ;
            end
            $\omega \leftarrow \omega \cdot \omega_m$ ;
        end
    end
end
```

powers. Since $\psi^{2k+1} = \omega^k \psi$, so we could use the standard NTT above on the scaled polynomial $a(\psi x)$, and we would obtain

$$c(\psi x) = INTT(NTT(a(\psi x)) \cdot NTT(b(\psi x)) .$$

This requires scaling of $a(x), b(x)$ and an inverse scaling of $c(\psi x)$, where the latter could be combined with the scaling by N in the final step of the INTT.

2.5 Optimizing the NTT

To optimize the NTT, it is possible to absorb the scaling by ψ and also to work around the BitReverse as is done in [9]. In the forward NTT, the function will return a ψ-scaled NTT in bit-reverse order, which will be undone in the INTT.

Let NTT_{sb} denote the function which computes the NTT of the scaled polynomial $a(\psi x)$ and where the output is given in bit-reversed order, in particular the output is given by

$$BitReverse([a(\psi\omega^k) : k \in [0 \ldots N-1]]) .$$

The resulting algorithm is given in Algorithm 2 and Ψ_{rev} is the array given by

$$\Psi_{rev} = BitReverse([\psi^k : k \in [0 \ldots N-1]]) .$$

Let $INTT_{sb}$ denote the function which computes the inverse NTT of a bit-reversed array including scaling by ψ^{-1}, i.e. $INTT_{sb}$ satisfies

$$INTT_{sb}(NTT_{sb}(a(x)) = a(x) .$$

To compute the product $c(x)$ of two polynomials in $a(x), b(x) \in \mathbb{Z}_q[x]/(x^N + 1)$, we can now simply compute it as

$$c(x) = INTT_{sb}(NTT_{sb}(a(x)) \cdot NTT_{sb}(b(x)) .$$

Algorithm 2: NTT_{sb}

```
Input: A vector $a = [a[0], \ldots, a[N-1]]$ of elements in $\mathbb{Z}_q$ in standard order and $2N$-th
       primitive root $\psi \in \mathbb{Z}_q$ of unity, and precomputed table $\Psi_{rev}$ containing the powers
       of $\psi$ in bit-reversed order
Output: $NTT_{sb}(a)$, i.e. bit-reversed NTT of scaled $a(\psi x)$
begin
    $k = 0$
    for $len = N/2; len > 0; len = len/2$ do
        for $start = 0; start < N; start = j + len$ do
            $S = \Psi_{rev}[++k]$
            for $j = start; j < start + len; ++j$ do
                $t = S * a[j + len] \bmod q$
                $a[j + len] = a[j] - t \bmod q$
                $a[j] = a[j] + t \bmod q$
            end
        end
    end
end
```

Algorithm 3: $INTT_{sb}$

```
Input: A vector $a = [a[0], \ldots, a[N-1]]$ of elements in $\mathbb{Z}_q$ in bit-reversed order and $2N$-th
       primitive root $\psi \in \mathbb{Z}_q$ of unity, and precomputed table $\Psi_{rev}$ containing the powers
       of $\psi$ in bit-reversed order
Output: $INTT_{sb}(a)$ in standard ordering
begin
    $k = N$
    for $len = 1; len < N; len = 2 * len$ do
        for $start = 0; start < N; start = j + len$ do
            $S = -\Psi_{rev}[--k]$
            for $j = start; j < start + len; ++j$ do
                $t = a[j]$
                $a[j] = t + a[j + len] \bmod q$
                $a[j + len] = S * (t - a[j + len]) \bmod q$
            end
        end
    end
    for $j = 0; j < N; ++j$ do
        $a[j] = a[j]/N$
    end
end
```

The function NTT_{sb} uses $(N/2)\log_2(N)$ multiplications modulo q and a total of $N\log_2(N)$ additions/subtractions modulo q. Due to the final scaling, the function $INTT_{sb}$ requires $(N/2)\log_2(N)+N$ multiplications modulo q and $N\log_2(N)$ additions/subtractions modulo q.

A polynomial product in $\mathbb{Z}_q[x]/(x^N+1)$ therefore requires $(3N/2)\log_2(N) + 2N$ multiplications and $3N\log_2(N)$ additions/subtractions. This should be compared to the N^2 multiplications and additions required for schoolbook multiplication.

Example: for $N = 256$ we thus require 3328 multiplications modulo q and 6144 additions modulo q compared to the 65536 multiplications/additions for the schoolbook approach.

2.6 NTT for 4-element word arrays

In Miden VM, a word consists of 4 field elements and the RAM is addressable by words, so we can read or write 4 field elements in one cycle. As such, it is interesting to develop an NTT routine which takes into account the word-addressable memory. We see two possibilities depending on the application:

4 parallel NTTs In many cases, e.g. in Dilithium, more than one NTT is required to be executed at the same time, e.g. in Dilithium, we require $k + \ell$ NTTs and k INTTs for $(k, \ell) = (4, 4), (6, 5), (8, 7)$, so we can easily execute 4 NTTs in parallel.

The idea is simply to take the 4 input N-vectors and pack them columwise in the 4 element words, i.e. the N words consist of

$$[A[k], B[k], C[k], D[k]]$$

where A, B, C, D are the 4 input vectors. Algorithm 2 remains exactly the same, with the assumption that all operations are now executed in a SIMD fashion.

Example: For $N = 256$ this would result in 2048 reads and 2048 writes of a word, 2048 SIMD-adds and 1024 SIMD-muls.

Single NTT for 4-element word arrays Inspired by the 2-element word memory in [12], we extend this method to 4 element words, by packing two consecutive words in the 2-element array into one 4-element word.

In particular, we simply store 4 consecutive elements in a logical array as a 4-element word.

Looking at the NTT however, we see that in one iteration elements of the form $a[j]$ and $a[j+len]$ are manipulated for $len = N/2, \ldots, 1$ where len is divided by 2 in each iteration. This indicates that the input array to the NTT should be ordered as follows:

$$[[A[2k], A[2k + N/2], A[2k + 1], A[2k + 1 + N/2]] \, .$$

Furthermore, since in the next iteration we still want to have that elements which are combined in the NTT butterfly are in the same word, we will have to read out a second word which contains the elements $len/2$ removed. This allows us to keep elements that are combined in the butterfly in the same word, i.e. in every iteration the elements saved will be of the form

$$[[A[2k], A[2k + len], A[2k + 1], A[2k + 1 + len]] \, .$$

The result is summarized in Algorithm 4. Note that the input has to be formatted as above, but that the output is a linearly organized array of quadruples of the bitreversed result, i.e. if B would be the output of NTT_{sb}, then the output of $NTT4_{sb}$ simply is

$$[B[4k], B[4k + 1], B[4k + 2], B[4k + 3]] \, .$$

A similar approach can be taken for the inverse NTT, see e.g. `ntt_4_256.py` in the Code subdirectory.

Example: We obtain the following numbers for the NTT, where adds and muls refer to single adds and muls (not SIMD versions). The inverse NTT requires an extra N Reads and Writes and Muls if the scaling by N is not folded into the final iteration of the main loop.

N	Reads	Writes	Muls	Adds
256	512	512	1024	2048
512	1152	1152	2304	4608
1024	2560	2560	5120	10240

A savings of $N/4$ reads and writes can be obtained by combining the $len = 2$ and $len = 1$ loops in Algorithm 4. If a single butterfly is implemented in 1 Miden cycle, then the total number of cycles equals Reads + Writes + Muls, since there are Muls butterflies, which also equals the number of Adds.

2.7 Native parameter sets

For the public key encryption and publicly re-randomizable commitment schemes we rely on arithmetic in a ring that is natively compatible with the prime $p = 2^{64} - 2^{32} + 1$, meaning that there is no need for additional modular reduction. This ring, which is inspired by the ThreeBears NIST standardization candidate [8], is defined as

$$R_p = \frac{\mathbb{Z}_p[X]}{(X^n + 1)}$$

but embeds a lattice of dimension $4n$ through the packing scheme

$$\mathbb{Z}_p = \frac{\mathbb{Z}[Y]}{(Y^4 - Y^2 + 1, Y - 2^{16})} \quad .$$

In our application scenario we will always use $n = 64$.

Multiplication in this ring can be achieved using NTTs of dimension n. Elements of this ring are short if the balanced expansion base 2^{16} of its coefficients are short. Specifically, if a, b, c, d are small (in absolute value) integers, then $a + b \cdot 2^{16} + c \cdot 2^{32} + d \cdot 2^{48}$ is a small element of $\mathbb{Z}_p$, and when all the coefficients of a polynomial $f(X) \in R_p$ have this form then it is short.

More concretely, we use the centered binomial distribution $\mathit{\Xi}$ with bounds -8 and 8 (both inclusive) and standard deviation $\sigma = 2$. However, $\mathit{\Xi}$ is a distribution of small integers; we also need field elements whose 16-bit chunks are distributed according to $\mathit{\Xi}$. To this end, define the distribution $\mathit{\Upsilon} = \sum_{i=0}^{3} \xi_i 2^{16i}$ with $\xi_i \sim \mathit{\Xi}$. Algorithms 5 and 6 specify one way to sample these distributions. Sampling short polynomials consists of sampling a coefficient vector of length $n = 64$ whose elements are distributed according to $\mathit{\Upsilon}$.

On top of the ring structure we use the module approach with dimension m. So specifically, there is an $m \times m$ matrix, along with $m \times 1$ and $1 \times m$ vectors whose elements belong to R_p. The parameter m depends on the security level.

Algorithm 4: $NTT4_{sb}$

```
Input: A vector of quadruples $a = [[A[2k], A[2k+N/2], A[2k+1], A[2k+1+N/2]]$ for
       $k = 0, \dots, N/4$ of elements in $\mathbb{Z}_q$ and $2N$-th primitive root $\psi \in \mathbb{Z}_q$ of unity, and
       precomputed table $\Psi_{rev}$ containing the powers of $\psi$ in bit-reversed order
Output: $NTT_{sb}(a)$, i.e. bit-reversed NTT of scaled $a(\psi x)$ in linear order as a vector of
        quadruples
begin
    $k = 0$
    for $len = N/2; len > 2; len = len/2$ do
        for $start = 0; start < N/4; start = j + len/2$ do
            $S = \Psi_{rev}[++k]$
            for $j = start; j < start + len/4; ++j$ do
                $A1 = a[j]$ // first quadruple
                $A2 = a[j + len/4]$ // second quadruple
                $t1 = S * A1[1]$
                $t2 = S * A2[1]$
                $t3 = S * A1[3]$
                $t4 = S * A2[3]$
                $a[j] = [A1[0] + t1, A2[0] + t2, A1[2] + t3, A2[2] + t4]$
                $a[j + len/4] := [A1[0] - t1, A2[0] - t2, A1[2] - t3, A2[2] - t4]$
            end
        end
    end
    // len = 2 done separately
    for $j = 0; j < N/4; ++j$ do
        $S = \Psi_{rev}[++k]$
        $A1 = a[j]$
        $t1 = S * A1[1]$
        $t2 = S * A1[3]$
        $a[j] := [A1[0] + t1, A1[2] + t2, A1[0] - t1, A1[2] - t2]$
    end
    // len = 1 done separately
    for $j = 0; j < N/4; ++j$ do
        $S = \Psi_{rev}[++k]$
        $A1 = a[j]$
        $t1 = S * A1[1]$
        $S = \Psi_{rev}[++k]$
        $t2 = S * A1[3]$
        $a[j] := [A1[0] + t1, A1[2] + t2, A1[0] - t1, A1[2] - t2]$
    end
end
```

Algorithm 5: sample_small_integer

```
Input:
Output: an integer distributed according to $\Xi$
begin
    $bits \xleftarrow{\$} \mathbb{Z}/2^{16}\mathbb{Z}$ // 16 random bits
    $num\_set \leftarrow 0$
    while $bits \neq 0$ do
        $bits \leftarrow bits \& (bits - 1)$ // bitwise and
        $num\_set \leftarrow num\_set + 1$
    end
    return $num\_set - 8$
end
```

sec. lvl	m
128	3
192	4
256	6

Algorithm 6: sample_short_field_element

```
Input:
Output: a field element distributed according to $\Upsilon$
begin
1   $acc \leftarrow 0$
2   for $i \in \{0, \ldots, 3\}$ do
3       $\xi \leftarrow$ sample_small_integer()
4       $acc \leftarrow acc \cdot 2^{16} + \xi$
5   end
6   return $acc$
end
```

3 Signatures

3.1 Dilithium

Dilithium is a module-LWE based signature scheme based on Fiat-Shamir with aborts. The base ring is given by $R_{q,n} = \mathbb{Z}_q[x]/(x^n + 1)$ with $q = 2^{23} - 2^{13} - 1 = 8380417$.

The public key consists of the high bits (13 lower order bits are dropped) of MLWE-samples

$$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$$

where $\mathbf{A}$ is a $k \times \ell$ matrix over $R_{q,n}$ derived from a seed ρ and $\mathbf{s}_1, \mathbf{s}_2$ vectors of length ℓ and k make up the secret key and contain elements in $R_{q,n}$ with small coefficients of max size η. A public key then contains the following data $pk = (\rho, \mathbf{t}_1)$ where $\mathbf{t}_1$ is obtained from $\mathbf{t}$ by dropping the lowest 13 bits. A summary of the relevant (for verification) parameters is given in Table 1.

Table 1. Dilithium parameters used in verification

Security level	2	3	5
(k, ℓ)	(4,4)	(6,5)	(8,7)
η	2	4	2
β	78	196	120
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
ω	80	55	75

A signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$ consists of 3 components:

- $\tilde{c}$: a challenge of 256 bits obtained as the hash of the public key, message and $\mathbf{Ay}$ where $\mathbf{y}$ is a vector with elements in $R_{q,n}$ with coefficients smaller than γ_1.
- $\mathbf{z}$: a vector of length ℓ of polynomials with small coefficients, in particular, smaller than $\gamma_1 - \beta$.

– **h**: a vector of length k consisting of k hint polynomials (essentially the overflows of a particular sum) whose coefficients are $0, 1$ and the max number of 1's in $\mathbf{h}$ is ω.

The signature verification then proceeds as follows, where H is SHAKE-256 and the specification of the other subroutines can be found in the original specification [10].

Algorithm 7: *Dilithium verification*

```
Input: public key $pk = (\rho, \mathbf{t}_1)$, message $M$, signature $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$
Output: boolean indicating if signature is valid
begin
    $\mathbf{A} \in R_{q,n}^{k\times\ell} :=$ ExpandA($\rho$)
    $\mu \in \{0,1\}^{512} := H(H(\rho||\mathbf{t}_1)||M)$
    $c := SampleInBall(\tilde{c})$
    $\mathbf{w}_1' := UseHint_q(h, \mathbf{Az} - 2^d \cdot c\mathbf{t}_1, 2\gamma_2)$
    return $||\mathbf{z}||_\infty < \gamma_1 - \beta$ and $\tilde{c} = H(\mu||\mathbf{w}_1')$ and $\#1's$ in $\mathbf{h}$ is $\leq \omega$
end
```

The matrix $\mathbf{A} \in R_{q,n}^{k\times\ell}$ is generated from ρ but directly into the NTT domain. In particular, every element $a_{i,j} \in R_q$ is represented in a very specific format, which is the following:

– Let $r = 1753$ which is a 512-th root of unity modulo q
– A polynomial $a(x) \in R_q$ in NTT representation then is given by the array:

$$[a(r_0), a(-r_0), a(r_1), a(-r_1), \ldots, a(r_{127}), a(-r_{127})]$$

where by definition $r_i = r^{brv(128+i)}$ where $brv(k)$ denotes the 8-bit bitreversal of the number k. Note that this is the same result as the bit-reversed version of the normal array

$$[a(r), a(r^3), \ldots, a(r^{N-1})]\,.$$

This is done to speed up step 4 in the verification procedure above, which can be computed as

$$NTT^{-1}(\mathbf{A} \cdot NTT(\mathbf{z}) - NTT(c) \cdot NTT(\mathbf{t}_1 \cdot 2^d))$$

which requires (*):

– $\ell + k + 1$ NTTs
– k inverse NTTs
– $k(\ell + 1)$ pointwise multiplications of arrays of 256 elements in $\mathbb{F}_q$

Integrating Dilithium in Miden Due to the formatting of $\mathbf{A}$, i.e. it is derived from ρ but directly into the NTT domain (with given ordering) as defined in Dilithium, it is impossible to directly use this $\mathbf{A}$ with the native Miden NTT routine.

If compatibility is required, then the only option is to first represent $\mathbf{A}$ in the Miden NTT domain, by first mapping $\mathbf{A}$ to the time domain and then mapping it to the Miden NTT domain. The result $\tilde{\mathbf{A}}$ is fully equivalent with $\mathbf{A}$, just represented in a different NTT domain. This transformation however is costly, in that it requires $k \times \ell$ calls to the original inverse NTT routine and also $k \times \ell$ calls to the Miden NTT routine. Of course, it is possible to consider $\tilde{\mathbf{A}}$ as part of the public key directly in Miden, but it is then impossible to directly derive it from ρ (without going through the different NTTs).

It compatibility is not required, i.e. it is allowed to derive a different public key from ρ, it looks tempting to derive $\tilde{\mathbf{A}}$ directly in the Miden NTT domain from ρ. However, this is not possible since the corresponding polynomials in the time domain should have coefficients uniform modulo the Dilithium modulus and in particular, they should be small, since otherwise we cannot execute the Dilithium operations by using the Miden arithmetic. As such, the only option here seems to be to use the same *ExpandA* routine as in Dilithium, but to consider the result to be defined in the time domain and then use $k \times \ell$ Miden NTTs to map it to the frequency domain. This approach will add another $k \times \ell$ Miden NTTs on top of (*) which makes it the dominating cost.

3.2 Falcon

Falcon [6] is a lattice-based signature scheme using NTRU lattices and a hash-and-sign approach following GPV [7]. Falcon works in a cyclotomic ring $R_{q,n} = \mathbb{Z}_q[x]/(x^n + 1)$ with $n = 2^k$, and an NTRU public key consists of a polynomial $h \in R_{q,n}$ which is computed as $g \cdot f^{-1}$ in $R_{q,n}$ where $f, g \in R_{q,n}$ that have small coefficients. Recovering f, g from h corresponds to the NTRU-problem.

Falcon specifies two parameter sets $n = 512$ and $n = 1024$ corresponding to NIST security levels I and V. The modulus $q = 12289 = 12 \cdot 1024 + 1$ is prime and fixed for both sets. The coefficients of the polynomials f, g are sampled from a discrete Gaussian with standardard deviation $\sigma_{f,g} = 1.17\sqrt{q/2n}$.

The signature of a message m consists of a salt r and a pair of small polynomials (s_1, s_2) such that $s_1 + s_2 h = H(r||m)$. Furthermore, s_1 can be derived fully from m, r and s_2 so the signature is simply given by (r, s_2). The signature (s_1, s_2) must satisfy $||(s_1, s_2)||^2 \leq \lfloor \beta^2 \rfloor$ where $\beta = 1.1\sigma\sqrt{2n}$. In particular, $\beta^2 = 34034726$ for Level-I parameters and $\beta^2 = 70265242$ for Level-V parameters. To verify a Falcon signature, one proceeds as described in Algorithm 8

Integrating Falcon in Miden Compared to Dilithium, Falcon is much easier to integrate into Miden due to the fact that the main operation in Algorithm 8 is step 4 which mainly consists of the multiplication $s_2 h$. Since both polynomials are given in the time domain, it is easy to compute their product by using the Miden NTT and inverse NTT routines as described in Section 2.4 at a cost of two NTTs and 1 inverse NTT plus n centered reductions modulo the Miden prime and n reductions modulo the Falcon prime.

Algorithm 8: *Falcon verification*

```
Input: Message $m$, signature $sig = (r, s)$, publick key $pk = h$ and bound $\lfloor \beta^2 \rfloor$
Output: boolean indicating if signature is valid
begin
1   $c := HashToPoint(r||m, q, n)$
2   $s_2 = Decompress(s, 8 \cdot sbytelen - 328)$
3   if $(s_2 = \perp)$ then return false
4   $s_1 = c - s_2 h \bmod q$
5   if $||(s_1, s_2)||^2 \leq \lfloor \beta^2 \rfloor$ then return true else return false
end
```

4 Encryption

This section specifies a public key encyption scheme over the native finite field $\mathbb{Z}_p$.

4.1 Basic Description

The scheme employs matrix multiplications where the matrix algebra is defined relative to the base ring $R_p = \frac{\mathbb{Z}_p[X]}{(X^n+1)}$ where $n = 64$. The elements of this base ring are polynomials, and so the elements of the matrices and vectors are polynomials. However, the fact that they are polynomials is only relevant to define the multiplication law, which can be computed using the fast NTT-based algorithm described in § 2.4. (The addition law is trivially element-wise addition.) It is natural and fitting to represent these polynomials as vectors of $n = 64$ field elements.

Let $\mathbf{G} \in R_p^{m\times m}$ be an arbitrary matrix, and let $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in R_p^{m\times 1}$ be vectors of short polynomials. The encryption scheme builds on the following noisy Diffie-Hellman protocol:

– The square matrix $\mathbf{G}$ is a public parameter known to both Alice and Bob.
– Alice samples $\mathbf{a}, \mathbf{b}$, computes $\mathbf{A} = \mathbf{G}\mathbf{a} + \mathbf{b}$, and sends $\mathbf{A}$ to Bob.
– Bob samples $\mathbf{c}, \mathbf{d}$, computes $\mathbf{B} = \mathbf{G}^\mathsf{T}\mathbf{c} + \mathbf{d}$, and sends $\mathbf{B}$ to Alice.
– Alice receives $\mathbf{B}$ from Bob and computes $K_A = \mathbf{a}^\mathsf{T}\mathbf{B}$.
– Bob receives $\mathbf{A}$ from Alice and computes $K_B = \mathbf{c}^\mathsf{T}\mathbf{A}$.

Alice's view K_A and Bob's view K_B of the shared secret key are close in the following sense. The difference $K_A - K_B = \mathbf{a}^\mathsf{T}\mathbf{d} - \mathbf{b}^\mathsf{T}\mathbf{c}$ is *short* – every coefficient of this polynomial has a balanced base-2^{16} expansion with a small ℓ_2-norm. Specifically, the variance of this ℓ_2-norm is $\sigma^2 \cdot \sqrt{8mn}$, where $\sigma = 2$ is the standard deviation in the distribution Ξ of § 2.7. For reference, for the worst case parameter set this value is roughly 222 whereas for *random* field elements it is roughly $\sqrt{\frac{2^{32}-1}{12}} \approx 18919$. Therefore, with high likelihood Alice and Bob will agree upon the top bit of all these 16-bit chunks.

The encryption scheme embeds the message in the top bit of each 16-bit chunk, and pads the resulting embedding with the noise shared one-time pad. With high likelihood, the noise does not disturb the message.

Algorithm 9: embed_msg

```
Input: a message $m \in \{0,1\}^{256}$
Output: a polynomial $M \in R_p$
begin
    return $[\sum_{i=0}^{3} 2^{16i+15} \cdot m[i+4j] : 0 \le j < 64]$
end
```

Algorithm 10: extract_msg

```
Input: a polynomial $M \in R_p$
Output: a message $m \in \{0,1\}^{256}$
begin
    $m \leftarrow []$
    for $c \in M$ do
        for $i \in \{0,3\}$ do
            $chunk \leftarrow c\&$0xffff
            $c \leftarrow c \gg 16$
            if $chunk < 2^{14} \vee 2^{16} - chunk < 2^{14}$ then
                $m \leftarrow m \| 0$
            end
            else
                $m \leftarrow m \| 1$
            end
        end
    end
    return $m$
end
```

4.2 Naive Scheme

In more detail, the public key encryption scheme consists of the following objects.

- The matrix $\mathbf{G} \in R_p^{m\times m}$ is a public parameter.
- A secret key is a pair of short vectors $\mathbf{a}, \mathbf{b} \in R_p^{m\times 1}$.
- A public key is single vector $\mathbf{A} \in R_p^{m\times 1}$.
- A message is a list of 256 bits $m \in \{0,1\}^{256}$.
- A ciphertext is a pair $(\mathbf{B}, C) \in R_p^{m\times 1} \times R_p$.

We start by defining a functionality that encryption and decryption relies on, namely the embedding and extraction of a message $m \in \{0,1\}^{256}$ into the top bits of 16-bit chunks of a polynomial $f \in R_p$.

The following algorithms (Algorithms 11-12-13) specify the public key encryption scheme. Note that the operations $+$ and $\times$ apply to vectors of 64 field elements. Specifically, these operations compute addition and multiplication in the ring R_p. This corresponds to element-wise addition and multiplication of polynomials followed by reduction modulo $X^{64}+1$.

4.3 Optimized Scheme

The strategy to compute multiplications in the polynomial quotient ring R_p by using NTTs followed by INTTs is redundant because almost all INTT maps at the end of one operation are follwed up with an NTT map preparing for the

Algorithm 11: KeyGen

```
Input:
Output: a secret key $sk$ and public key $pk$
begin
    $\mathbf{a} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \leq i < 64] : 0 \leq j < m]$
    $\mathbf{b} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \leq i < 64] : 0 \leq j < m]$
    // compute Alice's Diffie-Hellman contribution
    $\mathbf{A} \leftarrow [[0 : 0 \leq i < 64] : 0 \leq j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \times \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $sk = (\mathbf{a}, \mathbf{b})$, $pk = \mathbf{A}$
end
```

Algorithm 12: Enc

```
Input: a public key $pk = \mathbf{A}$, a message $m \in \{0,1\}^{256}$
Output: a ciphertext $ctxt = (\mathbf{B}, C)$
begin
    $\mathbf{c} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \leq i < 64] : 0 \leq j < m]$
    $\mathbf{d} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \leq i < 64] : 0 \leq j < m]$
    // compute Bob's Diffie-Hellman contribution
    $\mathbf{B} \leftarrow [[0 : 0 \leq i < 64] : 0 \leq j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{G}[j][i] \times \mathbf{c}[j]$
        end
        $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{d}[i]$
    end
    // compute shared noisy one-time pad
    $K \leftarrow [\text{sample\_short\_field\_element}() : 0 \leq i < 64]$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K + \mathbf{c}[i] \times \mathbf{A}[i]$
    end
    // pad message
    $C \leftarrow K + \text{embed\_msg}(m)$
    return $ctxt = (\mathbf{B}, C)$
end
```

Algorithm 13: Dec

```
Input: a secret key $sk = (\mathbf{a}, \mathbf{b})$, a ciphertext $ctxt = (\mathbf{B}, C)$
Output: a message $m \in \{0,1\}^{256}$
begin
    // compute shared noisy one-time pad
    $K \leftarrow [0 : 0 \leq i < 64]$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K + \mathbf{B}[i] \times \mathbf{a}[i]$
    end
    // unpad message
    $M \leftarrow C - K$
    return $\text{extract\_msg}(M)$
end
```

next operation. It therefore pays to represent the polynomials in the frequency domain. The time domain representation is only necessary when sampling small

Algorithm 14: KeyGen *(optimized)*

```
Input:
Output: a secret key $sk$ and public key $pk$, both represented in frequency domain
begin
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    $\mathbf{b} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{a}[i])$
        $\mathsf{NTT}_{sb}(\mathbf{b}[i])$
    end
    // compute Alice's Diffie-Hellman contribution
    $\mathbf{A} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \circ \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $sk = (\mathbf{a}, \mathbf{b})$, $pk = \mathbf{A}$
end
```

Algorithm 15: Enc *(optimized)*

```
Input: a public key $pk = \mathbf{A}$ represented in frequency domain, a message $m \in \{0,1\}^{256}$
Output: a ciphertext $ctxt = (\mathbf{B}, C)$ represented in frequency domain
begin
    $\mathbf{c} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    $\mathbf{d} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{c}[i])$
        $\mathsf{NTT}_{sb}(\mathbf{d}[i])$
    end
    // compute Bob's Diffie-Hellman contribution
    $\mathbf{B} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{G}[j][i] \circ \mathbf{c}[j]$
        end
        $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{d}[i]$
    end
    // compute shared noisy one-time pad
    $K \leftarrow [\mathsf{sample\_short\_field\_element}() : 0 \le i < 64]$
    $\mathsf{NTT}_{sb}(K)$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K + \mathbf{c}[i] \circ \mathbf{A}[i]$
    end
    // embed and pad message
    $M \leftarrow \mathsf{embed\_msg}(m)$
    $\mathsf{NTT}_{sb}(M)$
    $C \leftarrow K + M$
    return $ctxt = (\mathbf{B}, C)$
end
```

elements and when decoding the message. This observation gives rise to the equivalent public key encryption scheme specified by Algorithms 14-15-16. Note that the secret key, public key, and ciphertext are now represented in frequency domain. We use $\circ$ to denote the Hadamard (element-wise) product.

Algorithm 16: Dec *(optimized)*

```
Input: a secret key $sk = (\mathbf{a}, \mathbf{b})$, a ciphertext $ctxt = (\mathbf{B}, C)$ all represented in frequency
       domain
Output: a message $m \in \{0,1\}^{256}$
begin
    // compute shared noisy one-time pad
    $K \leftarrow [0 : 0 \leq i < 64]$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K + \mathbf{B}[i] \circ \mathbf{a}[i]$
    end
    // unpad, intt, and extract message
    $M \leftarrow C - K$
    $\mathsf{INTT}_{\mathsf{sb}}(M)$
    return $\mathsf{extract\_msg}(M)$
end
```

4.4 Security, Parameters

The encryption scheme is $\mathsf{IND-CPA}$-secure under the assumed hardness of a specific member of the decisional Ideal LWE class of problems as described by Bootland *et al.* [2]. This specific member is defined below.

Hard Problem. Let R_p and Υ be defined as in § 2.7. Let k, l, m be small integers. The $\mathrm{IMLWE}^*_{k,l,m}$ is to distinguish the distribution (1) from the distribution (2) given the sample $(\mathbf{A}, \mathbf{B}) \in R_p^{k \times l} \times R_p^{k \times m}$ where

(1) $\mathbf{A}$ is uniformly random and $\mathbf{B} = \mathbf{Ac} + \mathbf{d}$ for some matrices $\mathbf{c} \sim \Upsilon^{l \times m}$ and $\mathbf{d} \sim \Upsilon^{k \times m}$;
(2) $\mathbf{A}$ and $\mathbf{B}$ are uniformly random.

The security reduction constructs a IMLWE solver from a $\mathsf{IND-CPA}$ adversary $\mathcal{A}$. The reduction proceeds in two steps.

In the first step, $\mathcal{A}$ is used to solve a Diffie-Hellman-like problem, where the task is to distinguish the distribution (1) $(\mathbf{G}, \mathbf{A}, \mathbf{B}, K)$ from (2) $(\mathbf{G}, \mathbf{A}, \mathbf{B}, U)$, where all symbols match with their use in the specification of the encryption scheme and where $U \sim \mathcal{U}(R_p)$. In fact, this step is trivial. Supply the adversary with the public parameter $\mathbf{G}$, the public key $\mathbf{A}$, and the ciphertext $(\mathbf{B}, C)$ were C was constructed according to the last lines of Enc. If $\mathcal{A}$ correctly identifies which message was encrypted, then the fourth element of the tuple is not uniform. Conversely, if $\mathcal{A}$'s can do no better than guess at random then the fourth element must be uniform.

In the second step, a solver for this Diffie-Hellman-like problem is used to build one for $\mathrm{IMLWE}^*_{m,1,1}$. This step is analogous to § 5.5 of Frodo [3].

To estimate the hardness of $\mathrm{IMLWE}^*_{m,1,1}$ we use the tools of Albrecht *et al.* [5] and Ducas *et al.* [4]. While these tools estimate the hardness of standard LWE instances of dimension N, we argue that they apply also to $\mathrm{IMLWE}^*_{m,1,1}$ with $N = 4mn$, where the factor 4 arises from the integer packing scheme pressing 4 integers into every field element, and the factor n arises from the ring R_p.

The aptitude of these estimators warrants a note of caution. The estimators work for a generic q-ary lattice. However, the lattice induced by $\mathrm{IMLWE}^*_{m,1,1}$ is not q-ary and has abundant structure corresponding to the algebra over which multiplication is defined. Nevertheless, inspection of the generating matrix shows that the lattice in question is *very close* to q-ary, with $q = 2^{16}$. Moreover, the same estimators are used to estimate the hardness of the *structured* (thus not generic) lattice problems underlying Ring- and Module-based lattice cryptosystems. Neither of these caveats are known to give rise to exploitable attacks or even different attack complexity.

Table 2. Parameters, security, failure probability

sec. lvl.	m	Albrecht *et al.*	Ducas *et al.*	failure probability
128	3	148.9	135.3	$< 2^{-995}$
192	4	211.7	192.2	$\sim 2^{-725}$
256	6	263.5	310.8	$\sim 2^{-331}$

4.5 Homomorphisms and Failure Probability

The encryption scheme admits two homomorphic operations. First, addition of ciphertexts corresponds to addition of plaintexts modulo 2. Second, multiplication by *sufficiently short* elements of R_p affects the underlying plaintexts in the same way. Both operations can lead to a decryption failure, even if the operand ciphertexts do not, although this event happens with small probability.

To compute an approximation of the probability of decryption failure after any number of homomorphic operations, it is necessary to represent various distributions on a single packed integer. To make this calculation feasible, it is advisable to restrict distributions to the range $[-2^{14}, 2^{14}]$ by truncation. The probability of decryption failure corresponds to the distance between 1 and the sum of all probabilities of integers in this range. Starting from the distribution of small integers Ξ, this distribution evolves as follows.

– Multiplication of integers corresponds to convolution of their probability distributions.
– Multiplication of field elements gives rise to at least one packed integer consisting of the sum of 8 products of original packed integers.
– Multiplication of polynomials in R_p generates a polynomial where each coefficient consists of the sum of two products of field elements.
– Multiplication of an $l \times m$ matrix of polynomials by an $m \times 1$ vector of polynomials generates an $l \times 1$ vector where each coordinate consists of the sum of m products of polynomials.

It is feasible to apply the same homomorphic circuit to the distribution of small elements. The failure probability of decryption of the output ciphertexts is approximately the distance of the sum of this distribution from 1.

5 Post-quantum commitments

This section specifies a publicly rerandomizable commitment scheme over the native finite field $\mathbb{Z}_p$. A basic commitment scheme consists of two functions:

- commit, takes a message and some randomness and outputs a *commitment* along with some *decommitment information*.
- verify, takes a commitment, a message, decommitment information, and outputs a bit indicating whether the commitment is valid.

A commitment scheme is *publicly rerandomizable* when third parties can derive a new commitment so that:

a) The new commitment is unlinkable to the original commitment except by the party that produced the original commitment or the party that derived the new one.
b) The party that produced the original commitment can open the new one as well as the old one, but only to the same message.

We build this functionality using the ring $R_p = \frac{\mathbb{Z}_p}{\langle X^n+1\rangle}$ and associated tools as follows. Let $\mathbf{G} \in R_p^{m\times m}$ be a pseudorandom $m \times m$ matrix consisting of polynomials, and let $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in R_p^{m\times 1}$ be vectors of short polynomials, and $e \in R_p$ a single short polynomial. Then $(\mathbf{G}, \mathbf{a}^\mathsf{T}\mathbf{G} + \mathbf{b}^\mathsf{T})$ is one MLWE sample, and $(\mathbf{Gc} + \mathbf{d}, \mathbf{a}^\mathsf{T}\mathbf{Gc} + \mathbf{b}^\mathsf{T}\mathbf{c} + e)$ is another. Both MLWE samples commit to $\mathbf{a}$, in the sense that it is a short approximate solution, and that such solutions are difficult to find. This observation gives rise to the following scheme:

- The matrix $\mathbf{G} \in R_p^{m\times m}$ is a public parameter.
- Cory the committer feeds the message $m \in \{0, 1\}^*$ into a pseudorandom number generator and uses it to sample $\mathbf{a}$ and $\mathbf{b}$.
- Cory computes the commitment as $\mathbf{a}^\mathsf{T}\mathbf{G} + \mathbf{b}^\mathsf{T}$.
- Rachel the rerandomizer samples $\mathbf{c}, \mathbf{d}, e$ and computes the re-randomized commitment as $(\mathbf{Gc} + \mathbf{d}, \mathbf{a}^\mathsf{T}\mathbf{Gc} + \mathbf{b}^\mathsf{T}\mathbf{c} + e)$.
- To open a commitment, Cory supplies m. From this message, $\mathbf{a}$ can be determined, and it can be verified to be a short approximate solution to the matching LWE sample.

5.1 Naive Scheme

The scheme consists of four functions, relative to a message space $\mathcal{M}$ and randomness $\mathcal{R}$, the latter of which doubles as the space of decommitment information. The commitment has a different data structure before and after rerandomization: before it is $R_p^{1\times m}$, whereas after it is $R_p^{m\times 1} \times R_p$.

- $\mathsf{Commit} : \mathcal{M} \times \mathcal{R} \rightarrow R_p^{1\times m} \times \mathcal{R}$
- $\mathsf{VerifyRaw} : \mathcal{M} \times \mathcal{R} \times {R_p}^{1\times m} \rightarrow \{\mathsf{True}, \mathsf{False}\}$
- $\mathsf{Rerandomize} : R_p^{1\times m} \rightarrow ({R_p}^{m\times 1} \times R_p)$

Algorithm 17: Commit

```
Input: a message $t \in \mathcal{M}$ and randomness $r \in \mathcal{R}$
Output: a commitment $\mathbf{A} \in R_p^{1\times m}$ and decommitment information
begin
    $uniform\_bytes \leftarrow \mathsf{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \ldots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \leq i < 64] : 0 \leq j < m]$
    $\mathbf{b} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j+2i : 128j+2(i+1)]) : 0 \leq i < 64] : m \leq j < 2m]$
    $\mathbf{A} \leftarrow [[0 : 0 \leq i < 64] : 0 \leq j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \times \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $commitment = \mathbf{A}$, $decommitment\_info = r$
end
```

Algorithm 18: Verify

```
Input: a message $t \in \mathcal{M}$, decommitment information $r \in \mathcal{R}$, a commitment $com \in R_p^{1\times m}$
Output: True or False
begin
    $uniform\_bytes \leftarrow \mathsf{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \ldots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \leq i < 64] : 0 \leq j < m]$
    $\mathbf{b} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j+2i : 128j+2(i+1)]) : 0 \leq i < 64] : m \leq j < 2m]$
    $\mathbf{A} \leftarrow [[0 : 0 \leq i < 64] : 0 \leq j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \times \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $com \overset{?}{=} \mathbf{A}$
end
```

– $\mathsf{VerifyRerandomized} : \mathcal{M} \times \mathcal{R} \times (R_p^{m\times 1} \times R_p) \to \{\mathsf{True}, \mathsf{False}\}$

In addition to this interface we need a pseudorandom mapping from message-randomness pairs to a short vector of polynomials. We construct this manually, starting from a cryptographically secure extendable output function (XOF) to sample uniform bytes, followed by a sampler to send these uniform bytes to short polynomials. We overload the function sample_short_field_element so that it can use the argument as random bits if it is supplied; otherwise the bits are sampled at random.

– $\mathsf{xof} : \mathcal{M} \times \mathcal{R} \times \mathbb{N} \to (\{0,1\}^8)^*$

Lastly, we need a procedure to test whether a given polynomial is short enough. To this end we recycle the extract_msg function. The polynomial is short enough if this function returns all zeros.

Algorithm 19: Rerandomize

```
Input: a commitment $\mathbf{A} \in R_p^{1\times m}$
Output: a rerandomized commitment $(\mathbf{B}, K) \in R_p^{m\times 1} \times R_p$
begin
    $\mathbf{c} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    $\mathbf{d} \leftarrow [[\text{sample\_short\_field\_element}() : 0 \le i < 64] : m \le j < 2m]$
    $\mathbf{B} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{G}[j][i] \times \mathbf{c}[j]$
        end
        $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{d}[i]$
    end
    $e \leftarrow [\text{sample\_short\_field\_element}() : 0 \le i < 64]$
    $K \leftarrow [0 : 0 \le i < 64]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $K \leftarrow K + \mathbf{A}[i] \times \mathbf{c}[i]$
        end
        $K \leftarrow K + e$
    end
    return $recom = (\mathbf{B}, K)$
end
```

Algorithm 20: VerifyRerandomized

```
Input: a message $t \in \mathcal{M}$, decommitment information $r \in \mathcal{R}$, and a rerandomized
       commitment $(\mathbf{B}, K) \in R_p^{m\times 1} \times R_p$
Output: True or False
begin
    $uniform\_bytes \leftarrow \text{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \ldots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\text{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K - \mathbf{a}[i] \times \mathbf{B}[i]$
    end
    return $\text{extract\_msg}(K) \overset{?}{=} 0^{256}$
end
```

5.2 Optimized Scheme

Like in the case of the encryption scheme, a lot of cycles are wasted going to and from frequency domain just to compute a multiplication. It is better to represent the relevant objects in frequency domain and map them to and from time domain only when needed. Specifically, NTTs are necessary after sampling short elements, and INTTs before testing the lengths of elements. This observation gives rise to the optimized variant of the scheme, whose algorithms follow.

5.3 Security, Parameters, and Failure Probability

The security analysis and failure probability analysis reduces to analyses done for the encryption scheme. As a result, we can reuse the same parameters for the same target security levels and achieve the same failure probabilities. The

Algorithm 21: Commit *(Optimized)*

```
Input: a message $t \in \mathcal{M}$ and randomness $r \in \mathcal{R}$
Output: a commitment $\mathbf{A} \in R_p^{1\times m}$ and decommitment information
begin
    $uniform\_bytes \leftarrow \mathsf{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \dots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \le i < 64] : 0 \le j < m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{a}[i])$
    end
    $\mathbf{b} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j{+}2i : 128j{+}2(i{+}1)]) : 0 \le i < 64] : m \le j < 2m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{b}[i])$
    end
    $\mathbf{A} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \dots, m-1\}$ do
        for $j \in \{0, \dots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \circ \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $commitment = \mathbf{A}$, $decommitment\_info = r$
end
```

Algorithm 22: VerifyRaw *(Optimized)*

```
Input: a message $t \in \mathcal{M}$, decommitment information $r \in \mathcal{R}$, a commitment $com \in R_p^{1\times m}$
Output: True or False
begin
    $uniform\_bytes \leftarrow \mathsf{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \dots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \le i < 64] : 0 \le j < m]$
    $\mathbf{b} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j{+}2i : 128j{+}2(i{+}1)]) : 0 \le i < 64] : m \le j < 2m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{a}[i])$
        $\mathsf{NTT}_{sb}(\mathbf{b}[i])$
    end
    $\mathbf{A} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \dots, m-1\}$ do
        for $j \in \{0, \dots, m-1\}$ do
            $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{G}[i][j] \circ \mathbf{a}[j]$
        end
        $\mathbf{A}[i] \leftarrow \mathbf{A}[i] + \mathbf{b}[i]$
    end
    return $com \overset{?}{=} \mathbf{A}$
end
```

table summarizing this is Table 2. What remains to be said here is why these properties reduce to prior analyses.

Correctness. Correctness of VerifyRaw follows from construction. Correctness of VerifyRerandomized is more intricate. This function returns False if the noise term

$$\mathbf{b}^\mathsf{T}\mathbf{c} + e - \mathbf{a}^\mathsf{T}\mathbf{d}$$

is larger than 2^{14} in any one packed digit of any coefficient. The probability of this event is (marginally) less than the probability of a decryption failure.

Algorithm 23: Rerandomize *(Optimized)*

```
Input: a commitment $\mathbf{A} \in R_p^{1\times m}$
Output: a rerandomized commitment $(\mathbf{B}, K) \in R_p^{m\times 1} \times R_p$
begin
    $\mathbf{c} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : 0 \le j < m]$
    $\mathbf{d} \leftarrow [[\mathsf{sample\_short\_field\_element}() : 0 \le i < 64] : m \le j < 2m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{c}[i])$
        $\mathsf{NTT}_{sb}(\mathbf{d}[i])$
    end
    $\mathbf{B} \leftarrow [[0 : 0 \le i < 64] : 0 \le j < m]$
    for $i \in \{0, \ldots, m-1\}$ do
        for $j \in \{0, \ldots, m-1\}$ do
            $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{G}[j][i] \circ \mathbf{c}[j]$
        end
        $\mathbf{B}[i] \leftarrow \mathbf{B}[i] + \mathbf{d}[i]$
    end
    $K \leftarrow [\mathsf{sample\_short\_field\_element}() : 0 \le i < 64]$ // initialize with $e \sim \Upsilon^{64}$
    $\mathsf{NTT}_{sb}(K)$
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K + \mathbf{A}[i] \circ \mathbf{c}[i]$
    end
    return $recom = (\mathbf{B}, K)$
end
```

Algorithm 24: VerifyRerandomized *(Optimized)*

```
Input: a message $t \in \mathcal{M}$, decommitment information $r \in \mathcal{R}$, and a rerandomized
       commitment $(\mathbf{B}, K) \in R_p^{m\times 1} \times R_p$
Output: True or False
begin
    $uniform\_bytes \leftarrow \mathsf{xof}(t, r, 256 \cdot m)$
    $ch \leftarrow [uniform\_bytes[128 \cdot i : 128 \cdot (i+1)] : i \in \{0, \ldots, 2m-1\}]$
    $\mathbf{a} \leftarrow [[\mathsf{sample\_short\_field\_element}(ch[128j + 2i : 128j + 2(i+1)]) : 0 \le i < 64] : 0 \le j < m]$
    for $0 \le i < m$ do
        $\mathsf{NTT}_{sb}(\mathbf{a}[i])$
    end
    for $i \in \{0, \ldots, m-1\}$ do
        $K \leftarrow K - \mathbf{a}[i] \circ \mathbf{B}[i]$
    end
    $\mathsf{INTT}_{sb}(K)$
    return $\mathsf{extract\_msg}(K) \stackrel{?}{=} 0^{256}$
end
```

Binding. The binding property decomposes into that of commitments before rerandomization and that after.

Before rerandomization: suppose a commitment $\mathbf{A}$ has two valid openings: (t_0, r_0) and (t_1, r_1). Let $(\mathbf{a_0}, \mathbf{b_0})$ and $(\mathbf{a_1}, \mathbf{b_1})$ be the pairs of short vectors of polynomials that arise after seeding (t_0, r_0) or (t_1, r_1) into the XOF and sampling short vectors of polynomials from the resulting output stream. Distinguish two cases:

– $(\mathbf{a_0}, \mathbf{b_0}) \neq (\mathbf{a_1}, \mathbf{b_1})$. Over the random coins of $(\mathbf{a_1}, \mathbf{b_1})$, the probability that $\mathbf{a_0}^\mathsf{T}\mathbf{G} + \mathbf{b_0}^\mathsf{T} = \mathbf{a_1}^\mathsf{T}\mathbf{G} + \mathbf{b_0}^\mathsf{T}$ is approximately $|R_p^m|^{-1} \approx 2^{-4096m}$. Therefore, the probability of sampling distinct short vectors in the same lattice from the XOF is negligible for adversaries with bounded time.

– $(\mathbf{a_0}, \mathbf{b_0}) = (\mathbf{a_1}, \mathbf{b_1})$. Each binomial coefficient has about 3.047 bits of entropy. Every field element has 4 binomial coefficients; every polynomial 64 field elements, and every vector m polynomials. The total is roughly $768m$ bits of entropy. The cost of finding a collision in this distribution is on the order of 2^{384m}.

Hiding. Ignore the cost of attacking the XOF. The attacker who obtains $(\mathbf{a}, \mathbf{b})$ from $\mathbf{A}$ can be used to undermine the security of the encryption scheme. The attacker who obtains $\mathbf{a}$ from $(\mathbf{B}, K)$ can likewise be used to undermine the security of the encryption scheme. Therefore, the hiding property of the commitment scheme is at least as strong as the IND-CPA of the encryption scheme.

Unlinkability. The adversary who can determine whether a pair $(\mathbf{A}, (\mathbf{B}, K))$ is matching or mismatching (i.e., fix the same short solution $(\mathbf{a}, \mathbf{b})$ or not) can be used to win the decisional Diffie-Hellman game. An analogous reduction to that of § 5.5 of the Frodo paper [3] reduces this adversary to a solver of $\mathrm{IMLWE}^*_{m,1,1}$. The hardness analysis of this problem is provided in Section 4.4.

Acknowledgments This project was made possible through the financial support of Polygon, for which the authors would like to express gratitude.

References

[1] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17.

[2] Carl Bootland, Wouter Castryck, Alan Szepieniec, and Frederik Vercauteren. Sok: On the security of cryptographic problems from linear algebra. 2021.

[3] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. 2016.

[4] Leo Ducas et al. A sage toolkit to attack and estimate the hardness of lwe with side information.

[5] Martin Albrecht et al. Security estimates for lattice problems.

[6] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru. *Submission to the NIST's post-quantum cryptography standardization process*, 2017. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[7] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. *Electron. Colloquium Comput. Complex.*, (133), 2007.

[8] Mike Hamburg. Post-quantum cryptography proposal: Threebears, 2017.

[9] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139, 2016.

[10] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. Crystals-dilithium. *Submission to the NIST's post-quantum cryptography standardization process*, 2017. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[11] Thomas Pornin. Ecgfp5: a specialized elliptic curve. Cryptology ePrint Archive, Paper 2022/274, 2022.

[12] Sujoy Sinha Roy, Furkan Turan, Kimmo Järvinen, Frederik Vercauteren, and Ingrid Verbauwhede. Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*, pages 387–398. IEEE, 2019.

[13] Robin Salen, Vijaykumar Singh, and Vladimir Soukharev. Security analysis of elliptic curves over sextic extension of small prime fields. Cryptology ePrint Archive, Paper 2022/277, 2022.